THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Selecting Heterogeneous Cores for Diversity

Citation for published version:
Tomusk, E-A, Dubach, C & O'Boyle, M 2016, 'Selecting Heterogeneous Cores for Diversity', *ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 4, pp. 1-25.
https://doi.org/10.1145/3014165

Digital Object Identifier (DOI):
10.1145/3014165

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
ACM Transactions on Architecture and Code Optimization

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. May. 2023

Selecting Heterogeneous Cores for Diversity [1] [2]

Erik Tomusk, University of Edinburgh
Christophe Dubach, University of Edinburgh
Michael O'Boyle, University of Edinburgh

Mobile devices with heterogeneous processors are becoming mainstream. With a heterogeneous processor, the runtime scheduler can pick the best CPU core for a given task based on program characteristics, performance requirements, and power limitations. For a heterogeneous processor to be effective, it must contain a diverse set of cores to match a range of runtime requirements and program behaviors. Selecting a diverse set of cores is, however, a non-trivial problem. Power and performance are dependent on both program features and the microarchitectural features of cores, and a selection of cores must satisfy the competing demands of different types of programs. We present a method of core selection that chooses cores at a range of power-performance points. Our algorithm is based on the observation that it is not necessary for a core to consistently have high performance or low power; one type of core can fulfill different roles for different types of programs. Given a power budget, cores selected with our method provide an average speedup of 6% on EEMBC mobile benchmarks, and a 24% speedup on SPECint 2006 benchmarks over the state of the art core selection method.

CCS Concepts: •**Computer systems organization** → **Heterogeneous (hybrid) systems;** *Multicore architectures;*

Additional Key Words and Phrases: heterogeneous, single-ISA, flexibility, diversity, power-aware, core selection, design space exploration

ACM Reference Format:
Erik Tomusk, Christophe Dubach, and Michael O'Boyle, 2016. Selecting Heterogeneous Cores for Diversity *ACM Trans. Architec. Code Optim.* 13, 4, Article 49 (December 2016), 25 pages.
DOI: 10.1145/3014165

1. INTRODUCTION

Single-ISA heterogeneity has attracted significant research interest over the past decade. Recently, mobile processors that implement two different CPU cores have entered the consumer market [Greenhalgh 2011]. The demand for heterogeneity in the mobile space is driven by the need for runtime power flexibility. Mobile devices cannot consume large amounts of power, yet users still desire high performance. A heterogeneous processor implements more than one type of core, and can provide the best performance within the amount of power that is available at a given time.

To design a heterogeneous processor, the designer must solve the *selection problem*—the designer must choose which different types of CPU cores should be implemented on the processor. The selection problem is distinct from design space exploration (DSE). DSE is used to find a potentially large set of cores that *could* be implemented. Core selection is used to choose which cores from DSE *should* be implemented. The selection problem has previously been discussed in detail [Tomusk et al. 2015a]. Some solutions have been proposed [Navada et al. 2013; Guevara et al. 2014]

[1] New paper, not an extension of a conference paper
[2] © 2016 Copyright held by the authors. This is the authors' version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in ACM TACO, http://doi.acm.org/10.1145/3014165.

Authors' addresses: E. Tomusk, C. Dubach, M. O'Boyle, School of Informatics, University of Edinburgh, 10 Crichton Street, EH8 9AB, United Kingdom; email: e.tomusk@ed.ac.uk, christophe.dubach@ed.ac.uk, mob@inf.ed.ac.uk.
2016 1544-3566/2016/12-ART49 $15.00
DOI: 10.1145/3014165

The selection problem is particularly difficult for mobile devices. A mobile processor must implement cores ranging from low power to high performance so that it can be fast when possible, and operate at low power when required. It is not sufficient to select cores that only maximize the performance of different types of programs, as done by Navada et al. [2013] for server processors; mobile devices also require low-power cores. The current state of the art in core selection chooses cores to maximize consistency—cores are selected to implement a range of power-performance points, but each core has a similar efficiency regardless of the type of task it is running [Guevara et al. 2014]. We find that enforcing consistency limits the selection unnecessarily, and in fact, the key to unlocking performance in a mobile, heterogeneous processor is to take advantage of the task-dependent performance variations of cores. Based on this insight, we define the *LUCIE* algorithm for selecting cores. LUCIE departs from previous work by taking a benchmark-centric approach to selection. Individual cores are not forced into specific roles, such as providing consistent efficiency or optimizing the performance of one benchmark. Instead, a small number of cores are selected such that the whole selection collectively provides a range of power-performance points for all benchmarks. Under a power budget, LUCIE leads to an average speedup of 6% on EEMBC benchmarks, and a 24% speedup on SPECint 2006 benchmarks, compared to the state of the art core selection method.

A thorough motivation for the selection problem is presented in section 2. Section 3 defines the LUCIE algorithm. Sections 4 and 5 describe our experiment infrastructure and evaluation methodology. Section 6 presents results from the LUCIE algorithm, and compares LUCIE to the state of the art selection technique. In section 7, we describe an extension to LUCIE for guaranteeing peak performance and for selecting cores to complement already implemented cores. The latter enables designing heterogeneous processors incrementally. In section 8, we demonstrate how an empirically determined available power distribution can be used to direct LUCIE. Related work is in section 9, and section 10 concludes.

2. MOTIVATING EXAMPLE

The key motivation for the use of heterogeneous processors in mobile devices is that a diverse set of cores enables flexibility at runtime [Tomusk et al. 2015a]. If the runtime scheduler has a range of cores to choose from, it can consider performance requirements, power constraints, and program characteristics, and schedule each task to the best core. A *task* could be an entire program, a program phase, or even a thread of a multithreaded program. The amount of flexibility available at runtime is determined by the types of cores that the processor designer chooses to implement at design time. The problem of core selection is therefore a problem of finding a small number of diverse cores that caters to a broad range of applications and operating conditions. In the following, we will first describe the selection problem in detail and then discuss existing solutions.

2.1. Problem Description

Selecting cores for mobile processors, such as those used in tablets and smartphones, presents unique challenges compared to selecting cores for desktops, servers, or microcontrollers. Mobile processors must perform well on many different types of programs while being constrained to strict power budgets. Since mobile devices depend on a battery and generally use passive cooling, the amount of power that a program can be allowed to consume is limited, and varies depending on battery charge, ambient conditions, and other running programs. Unlike a server, a modern eight-core mobile device is likely to be under-subscribed, and the user will be much more interested in turnaround time than system throughput.

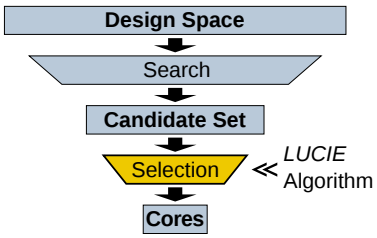

Fig. 1. The LUCIE algorithm selects the specified number of heterogeneous cores from a large candidate set.

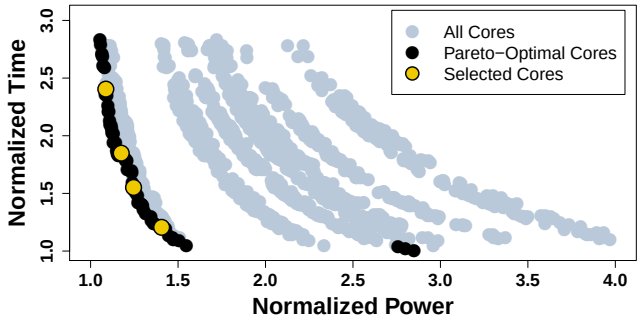

Fig. 2. All cores, the candidate set (Pareto-optimal cores), and selected cores, shown for the *aes* benchmark. Smaller is better on the axes.

We assume that a heterogeneous device has the following runtime behavior: The operating system scheduler determines that a program must be run. Using various metrics, such as the amount of power being consumed by the processor, the amount of remaining battery charge, the temperature of the device, the priority of the program, etc., the scheduler determines how much power can be allocated to the program. The scheduler then runs the program on the fastest core that can operate within the available power budget. The specifics of power-aware schedulers are beyond the scope of this paper, but recent work has demonstrated the possibility of adding this functionality to Linux [Panneerselvam and Swift 2016]. We wish to study the effects of the selected cores independently of a scheduler and any *uncore* components (e.g., network-on-chip, L2 cache). We therefore assume an oracle scheduler and that uncore components do not reduce the benefits provided by the heterogeneous set of cores.

Based on this description, it is obvious that greater microarchitectural diversity in the set of cores provides the scheduler with greater flexibility. Ideally, a processor would contain many cores that are tuned to different types of programs and that range from low power to high performance. This would allow the scheduler to maximize performance for all types of programs under different power budgets. However, a maximally diverse processor might contain hundreds of cores, while a real processor can only implement a few different types of cores. Core selection is therefore a problem of capturing the large amount of diversity available in a microarchitectural design space using only a small number of cores.

We illustrate how the core selection problem fits in with a broader heterogeneous design flow using figure 1. The design space of possible heterogeneous cores can contain billions of different cores, since a core is defined by a large number of parameters, and each parameter can take many values. The space must be searched for the best cores—the *candidate set*. This search step is often called DSE (design space exploration). Finally, some of the cores in the candidate set must be selected for inclusion

in a processor. Figure 2 shows an example of this process using the *aes* benchmark. The entire design space is shown with gray points (see section 4.2 for details on the design space). Most of these cores have such poor power and performance characteristics that they should never be considered for implementation. DSE is used to search the space for the candidate set of power-performance Pareto-optimal cores (black points). Any core in the candidate set could be implemented, but since there are tens of candidate cores, not all of them can be. A selection process must be used to choose a small number of candidate cores for implementation. Selection is made more difficult by the fact that the power and performance of a core is dependent on how a program interacts with a core's microarchitecture. Some cores will be Pareto-optimal for one type of program, but not for another. Some cores will be Pareto-optimal for many programs, but the exact power and performance will vary by program. For the remainder of the paper, we assume that the processor designer has access to a set of benchmarks that are representative of the types of programs that will be run on the processor.

2.2. Existing Solutions

There has been extensive work on design space exploration for heterogeneous processors [Lee and Brooks 2006; Karkhanis and Smith 2007; Lee and Brooks 2007; Kang and Kumar 2008; Azizi et al. 2010; Liu et al. 2011; Sunwoo et al. 2013; Turakhia et al. 2013]. These works focus on efficiently performing the search step in figure 1. Characterizing cores often requires substantial amounts of slow, cycle-accurate simulation. Since most of the cores in the design space are not Pareto-optimal, the time spent simulating them is ultimately a waste. DSE methods intelligently focus the simulation effort on the Pareto-optimal cores, thereby speeding up the process.

There has, however, been very little work on determining which of the Pareto-optimal cores found through DSE should be implemented. A combined DSE and selection method is proposed by Navada et al. [2013], but this method only optimizes for execution speed, and is therefore not appropriate for mobile devices. The state of the art in heterogeneous core selection is by Guevara et al. [2014]. We will refer to this as the *Clustering* selection method. The Clustering method groups similar cores together using k-means clustering, and then selects a representative core from each cluster. This leads to low-power cores, high-performance cores, and cores in between, as required by mobile processors. The processor designer defines how many different types of cores are needed by setting the number of clusters that k-means should produce. The remainder of this paper will describe the LUCIE algorithm for core selection, and will compare it to the Clustering selection method.

3. LUCIE ALGORITHM

LUCIE—the *L*east *U*seful *C*onfiguration *I*terative *E*limination algorithm—is a design-time algorithm that takes a candidate set of cores and removes cores from the set one at a time until the desired number of cores remains. The candidate set comes from a design space search (see figure 1). It contains all cores found during the search that are power– and performance– Pareto-optimal for at least one benchmark. The LUCIE algorithm has two related goals: to select cores at a variety of power-performance points ranging from low power to high performance, and to ensure that different types of programs (different benchmarks) have access to a range of power-performance points. While benchmarks are normally programs or program phases, a designer could make one program into several benchmarks by running the program with different inputs if the inputs have a large effect on program behavior. Since the power and performance of a core are dependent on the characteristics of the program being run, it is crucial that a selection algorithm considers how each of the representative benchmarks performs on each core. A core that is fast and power-hungry for one benchmark might be

slow, yet power-hungry for another. LUCIE also attempts to avoid disproportionately high-power and disproportionately slow cores. The fastest Pareto-optimal cores in figure 2, for example, consume a substantial amount of power but deliver an insignificant speed improvement.

The underlying contribution of the LUCIE algorithm is recognizing that one core can fulfill different roles for different types of benchmarks. A core might provide relatively high performance at a high power cost to one benchmark, it might provide moderate performance at a moderate cost to another benchmark, and it might not even be Pareto-optimal for a third benchmark. LUCIE selects cores based on the significance of their contribution to each benchmark, as compared to the significance of other cores. This is in contrast to the existing Clustering selection method, which first classifies cores as generally low-power, generally high-performance, etc., and then selects a core from each class. Both LUCIE and the Clustering method assume that the benchmarks are representative of the types of programs that will be run on the processor. In the following, we define the LUCIE algorithm and provide an example of its behavior.

3.1. Definition

LUCIE performs core selection in a normalized, unit-less metric space. Power and execution time are expressed as multiples of lowest per-benchmark power and shortest per-benchmark execution time to account for variations among benchmarks. We first provide details on the normalization process and then define the core elimination mechanism.

3.1.1. Metric Normalization. For each core-benchmark combination, power and execution time are divided by the best power and the best execution time for that benchmark by any core in the design space, as shown in equation 1. $P_{c,b}$ is the normalized (unit-less) power of core c executing benchmark b. $\mathrm{P_{raw}}(c,b)$ is the power consumption of b on c in Watts, as reported by a power model. The denominator divides $\mathrm{P_{raw}}$ by the lowest power achievable for the benchmark by any core in the Pareto-optimal set (candidate set), $\mathcal{C}$. Normalization is identical for time (T), where $\mathrm{T_{raw}}$ comes from a simulator and is measured in seconds. The normalized power and time values are computed only once.

$$P_{c,b} = \frac{\mathrm{P_{raw}}(c,b)}{\min\left(\mathrm{P_{raw}}(\mathcal{C},b)\right)} \qquad T_{c,b} = \frac{\mathrm{T_{raw}}(c,b)}{\min\left(\mathrm{T_{raw}}(\mathcal{C},b)\right)} \tag{1}$$

Power and time are normalized to enable fair comparisons between cores. As an example, knowing that benchmark *A* draws 500mW on a core and that benchmark *B* draws 1W on the same core is not particularly informative. Knowing that *A* runs at $2.0\times$ its minimum power and *B* runs at $1.1\times$ its minimum power on the core suggests that the core has good power characteristics for *B* but not for *A*, despite the fact that the absolute power for *B* is greater than for *A*. The remainder of the paper will use normalized power and time instead of absolute values. *Average normalized power* and *average normalized time* summarize the normalized power and time of a given core across all benchmarks using the arithmetic mean.

3.1.2. Configuration Elimination. LUCIE is shown in algorithm 1. The algorithm begins with the candidate set of cores, $\mathcal{C}$, where every core in $\mathcal{C}$ is power-performance Pareto-optimal for at least one benchmark. The cores in $\mathcal{C}$ come from a design space search (see figure 1). N is the number of different cores LUCIE should select. It is set by the designer. Each core in $\mathcal{C}$ has a list of one or more benchmarks for which it is Pareto-optimal. We refer to the length of this list as *affinity*—if a core is optimal for many

ALGORITHM 1: LUCIE configuration elimination

while $|\mathcal{C}| > N$ **do**
  forall the c *in* $\mathcal{C}$ **do**
    $\underline{\mathbf{C}}(c)$ ▷ Equation 2
  end
  $c \leftarrow$ FINDMINCOSTCORE($\mathcal{C}$)
  remove c from $\mathcal{C}$
  forall the b *in* $\mathcal{B}_c$ **do**
    $cm \leftarrow \arg\min_k \ \underline{\mathbf{BD}}(c, k, b)$ ▷ Equation 5
    append b to list for cm if not present
    $m_{cm,b} \leftarrow m_{cm,b} + m_{c,b} + 1$
  end
end

benchmarks, then it has a high affinity, and if it is optimal for a few, then it has a low affinity. Each benchmark in each core's affinity list also has a *move counter*, $m_{c,b}$, which has an initial value of 0. LUCIE first iterates over all cores in $\mathcal{C}$, and calculates each core's *cost* (defined below). It then finds the core with the minimum cost, c, and removes it from set $\mathcal{C}$. Finally, it iterates over all benchmarks that were associated with core c—the set $\mathcal{B}_c$. For each benchmark, b, in $\mathcal{B}_c$, it finds a destination core, cm. If b is already associated with cm, then the benchmark list for cm is unchanged. Otherwise, b is added to the list for cm, and the affinity of cm increases. cm is the core nearest to c for benchmark b based on our distance metric (see below). For each benchmark in $\mathcal{B}_c$, the move counter, $m_{c,b}$, is incremented and added to the destination core's move counter. The process repeats until the desired number of cores remain.

The cost of core c, $\underline{\mathbf{C}}(c)$, is given in equation 2. For each benchmark, b, associated with core c, we calculated $\underline{\mathbf{D}}$, the cost of displacing the benchmark from c. The total cost of core c is the sum of the individual displacement costs. The displacement cost is weighted by the move counter, $m_{c,b}$, which tracks how many times benchmark b has already been displaced. The move counter helps spread out the selection of cores. A core can have a high cost if there are many benchmarks associated with it, if a few benchmarks have a large displacement cost (i.e., if the core is very important to a few benchmarks), or if a benchmark that has already been displaced many times is associated with the core. In this default formulation, all benchmarks are assumed to be equally important. If the relative importance of benchmarks is known, then a further per-benchmark weighting term can be added to equation 2.

$$\underline{\mathbf{C}}(c) = \sum_{b}^{\mathcal{B}_c} \underline{\mathbf{D}}(c, b) \times (m_{c,b} + 1) \tag{2}$$

Equation 3 defines the displacement cost, $\underline{\mathbf{D}}(c, b)$. This is a measure of how useful core c is to benchmark b. If $\underline{\mathbf{D}}$ is large, then core c occupies a unique position in the design space for the benchmark. If it is small, then there exists another core that has a similar power-performance trade-off for b. If core c is to be removed, b should be displaced to the nearest core. We use a biased distance metric, $\underline{\mathbf{BD}}$, to find the distance between two cores for benchmark b. The closest core to core c for benchmark b is defined as core cm. The displacement cost is the cost of moving benchmark b from c to cm.

$$\begin{aligned} \underline{\mathbf{D}}(c,b) &= \underline{\mathbf{BD}}(c,cm,b) \\ cm &= \arg\min_{k} \underline{\mathbf{BD}}(c,k,b) \end{aligned} \tag{3}$$

Finally, we define the biased distance, $\underline{\mathbf{BD}}$, between core c_1 and core c_2 for benchmark b. Since no two benchmarks have exactly the same power-performance behavior, each benchmark has a different $\underline{\mathbf{BD}}$ for the same pair of cores. We first define the quantities $\Delta\mathrm{P}$ and $\Delta\mathrm{T}$ in equation 4. $\Delta\mathrm{P}$ is the difference between core c_1 and core c_2 for benchmark b along the normalized power axis; $\Delta\mathrm{T}$ is the difference along the normalized time axis. A positive $\Delta\mathrm{P}$ or $\Delta\mathrm{T}$ means that moving from c_1 to c_2 is an improvement for b. Since the cores in $\mathcal{C}$ follow a Pareto frontier, a positive $\Delta\mathrm{P}$ will generally be paired with a negative $\Delta\mathrm{T}$, and vice-versa.

$$\begin{aligned} \Delta\mathrm{P}(c_1,c_2,b) &= P_{\mathrm{c_1,b}} - P_{\mathrm{c_2,b}} \\ \Delta\mathrm{T}(c_1,c_2,b) &= T_{\mathrm{c_1,b}} - T_{\mathrm{c_2,b}} \end{aligned} \tag{4}$$

Equation 5 defines biased distance. $\underline{\mathbf{BD}}$ is the Euclidean distance between core c_1 and core c_2, multiplied by a biasing factor. If c_2 has both a higher power and a higher time than c_1 (negative Δ), then the biasing factor will be greater than one, and the cost of moving b from c_1 to c_2 will increase. If the move from c_1 to c_2 greatly reduces one metric at a small detriment to the other, then the biasing factor reduces the cost to favor the move. The effect of the biasing factor is negligible when the Δ values are of similar magnitude.

$$\underline{\mathbf{BD}}(c_1,c_2,b) = \sqrt{\Delta\mathrm{P}(c_1,c_2,b)^2 + \Delta\mathrm{T}(c_1,c_2,b)^2} \;\times\; \Big(1 - \big(\Delta\mathrm{P}(c_1,c_2,b) + \Delta\mathrm{T}(c_1,c_2,b)\big)\Big) \tag{5}$$

The biasing factor in equation 5 has two purposes. Since $\mathcal{C}$ contains Pareto-optimal cores for all benchmarks, the biasing factor discourages displacing a given benchmark to a nearby core that is not Pareto-optimal for the benchmark. The biasing factor also directs LUCIE away from extreme cores—cores that trade substantial increases in power for marginal performance increases, and cores that trade marginal power reductions for substantial performance reductions.

3.2. Example

Figure 3 shows the progression of the LUCIE algorithm as it eliminates cores. This example uses results from the 12 SPEC 2006 integer benchmarks (see section 4.1). The candidate set in this case contains 266 different cores. Cores are plotted by normalized power and performance, averaged across all benchmarks. Since averages hide per-benchmark variations, we use gray boxes to show the full range of power and performance values across all benchmarks for the final four cores. This illustrates that a core's behavior is dependent on the type of program being run.

The candidate set contains two clusters of cores that clearly fall onto the power-performance Pareto frontier, and an additional cluster of high-power cores. The high-power cores achieve a short execution time (are Pareto-optimal) for only a few benchmarks, and their affinity is therefore low. Since figure 3 is averaged across all benchmarks, the high-power cores appear above the Pareto frontier. These fast cores are undesirable for two reason: First, there are cores with similar speed characteristics that consume much less power. Second, a heterogeneous processor can only implement a limited number of core types, and in general, it is therefore better for cores to be

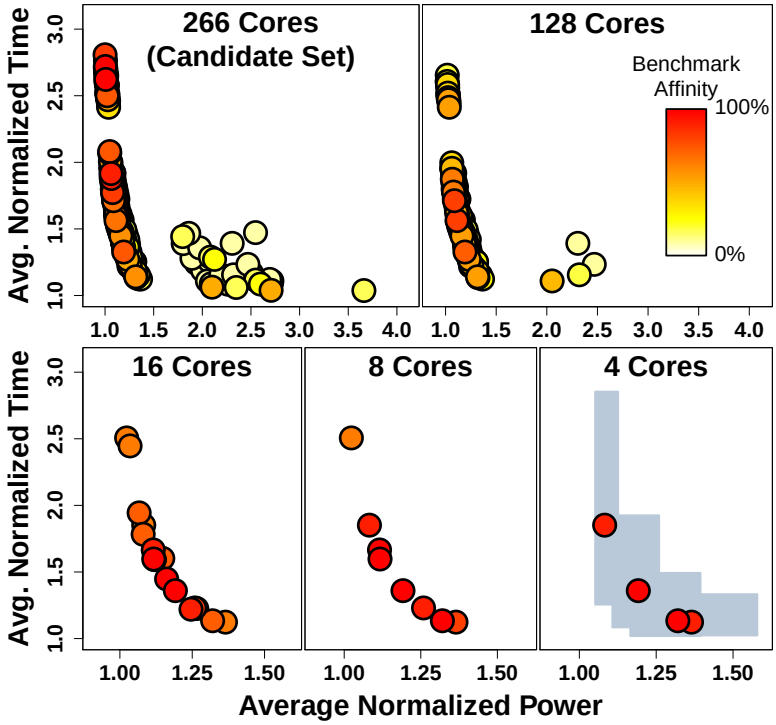

Fig. 3. LUCIE approximates the candidate set with an ever decreasing number of cores. Gray boxes show the full power and performance range of the final four cores.

broadly applicable. Equation 2 and the biasing factor in equation 5 ensure that the low-affinity cores have a low cost. The plot of 128 selected cores shows that LUCIE quickly eliminates most of them. We note that in some cases, an unusually high-power core may be required. E.g., there may be an important, outlier program that must be run very quickly. We will return to this issue in section 7.

Figure 3 shows that LUCIE removes the disproportionately high-power and disproportionately slow cores. The four selected cores approximate the knee of the original Pareto frontier. If eight cores are selected, then a medium-affinity core with very low power consumption is included, and there are more cores in the knee.

LUCIE approximates the full, Pareto-optimal set with a small number of cores, while balancing the competing demands of covering the design space and per-benchmark specialization. When four cores are selected, the cores are effectively general-purpose. They all have a high affinity, and any benchmark could be run on any core, dependent only on runtime power and performance requirements. When the number of cores increases to eight and beyond, some low-affinity cores appear. These provide significant benefits to only a few benchmarks—i.e., the cores are specialized, and each should only ever run a subset of tasks. LUCIE does not have a stopping criterion, but can be run until only one core remains. It is up to the designer to decide how many core types should be implemented, as this will depend on the specific design space, the target benchmarks, available engineering effort, etc. The designer might, for example, use LUCIE to eliminate cores from a candidate set until the remaining cores fit into the available area budget of the processor. Care must be taken when deciding whether to implement cores with lower affinities, since these benefit only a few benchmarks and represent a smaller return on engineering effort than high-affinity cores.

Table I. SPECint 2006 benchmarks

SPECint 2006	
perlbench	sjeng
bzip2	libquantum
gcc	h264ref
mcf	omnetpp
gobmk	astar
hmmer	xalancbmk

Table II. EEMBC benchmarks

DENBench (cryptography)			
aes	huffde	des	rsa
DENBench (audio-video)			
cjpegv2	mpeg2decode	mpeg4encode	rgbyiqv2
djpegv2	mpeg2encode	rgbcmykv2	-
mp3player	mpeg4decode	rgbhpgv2	-
Networking 2.0			
ip_pktcheck	nat	qos	tcp
ip_reassembly	ospfv2	routelookup	-

4. EXPERIMENT METHODOLOGY

We evaluate the LUCIE and Clustering algorithms on candidate sets of cores extracted from an example design space. Benchmarks, the design space, and the candidate sets are described below.

4.1. Benchmarks

We use the SPEC 2006 integer benchmark suite (table I) and the EEMBC DENBench (digital entertainment) and EEMBC Networking 2.0 suites (table II, see Poovey et al. [2009]). The EEMBC suites contain smaller benchmarks that represent common tasks on mobile devices. The SPEC suite contains more demanding benchmarks. All benchmarks are compiled for the ARM ISA using gcc with the `-O3` option. For SPEC benchmarks, we use the training data sets. We fast-forward the first two billion instructions, warm up for ten million CPU cycles, and perform detailed simulation for one billion instructions. The SPEC benchmarks always use an 8-way, 2MB L2 cache. For the much smaller EEMBC benchmarks, we perform detailed simulation from the beginning for one billion instructions or until completion, whichever comes first. We use the gem5 simulator with the cycle-accurate "arm_detailed" out-of-order core model [Binkert et al. 2011] and the McPAT power model [Li et al. 2009].

4.2. Design Space

Both LUCIE and the Clustering algorithm select cores based on observable features—power, performance, and efficiency—rather than the implementation details of the cores. Consequently, the design space must contain cores that operate at a range of power and performance levels, and at a range of efficiencies. The implementation details of the cores, however, are only relevant to an analysis of the algorithms (see section 6.1), and not to the algorithms themselves.

The example microarchitectural design space used to demonstrate the algorithms is shown in table III. Recent work has raised substantial concerns regarding McPAT's CPU core models, as pipeline logic is generally not modeled, the cost of storage is overestimated, and implicit fudge factors are used [Xi et al. 2015]. Despite these limitations, McPAT continues to be the state of the art general purpose power model. LUCIE naturally has some robustness against McPAT's limitations due to the use of a normalized metric space (section 3.1.1)—LUCIE assumes that power and performance values are accurate in relative but not necessarily in absolute terms. We further isolate our evaluation from these limitations by using only a subset of the microarchitectural design space exposed by gem5 and McPAT. The design space only varies cache-like structures, since McPAT models these in detail with CACTI. We specifically avoid crossing fudge factor boundaries in the design space. For example, when McPAT models in-order cores, it assumes that ALUs and other functional units have no base energy cost, even though ALUs are not related to issue logic. A design space composed entirely of in-order cores might be internally consistent in relative terms, but we have no con-

Table III. Summary of microarchitectural parameter values that are varied in the example design space

Parameter		Values	Parameter		Values
Data Cache	Size	16kB – 64kB	Branch Predictor	Global Counter Bits	1 – 3
	Ways	1 – 4		Global Entries	2^{10} – 2^{14}
Instruction Cache	Size	4kB – 64kB		Local Counter Bits	1 – 3
	Ways	1 – 4		Local History Bits	10 – 12
Registers	Integer	50 – 256		Local History Entries	2^{9} – 2^{11}
	Floating Point	96 – 256		Choice Counter Bits	1 – 3
Queue Entries	Issue	16 – 64		Choice Entries	2^{10} – 2^{14}
	Load	8 – 64	Branch Target Buffer	Entries	2^{10} – 2^{13}
	Store	8 – 64		Tag Bits	16 – 20
	Reorder Buffer	16 – 128	-	-	-

fidence that McPAT's in-order models are comparable to its out-of-order models. The out-of-order space is larger, and we use this for our design space.

We have also not included DVFS (dynamic voltage and frequency scaling) in the design space. This is partly again due to modeling limitations in McPAT. From first principles, one would expect that cores designed for higher frequencies would have a higher energy-per-instruction (EPI) than cores designed for lower frequencies, as faster clock rates require faster, more complex circuits. Applying voltage and frequency scaling to McPAT models leads to the opposite result: A core designed for 1GHz but run at 500MHz, for example, can reduce EPI by 40% compared to the same core designed and run at 500MHz. This is because DVFS has a large effect on pipeline logic that is not modeled by McPAT. As a result, it is difficult to draw meaningful conclusions about DVFS from McPAT. Furthermore, the future of DVFS remains unclear. Lukefahr et al. [2014] show that heterogeneity is already more efficient than DVFS. As process technology continues to scale, DVFS is expected to become less and less effective due to a decreasing range of usable voltages [Le Sueur and Heiser 2010; Etinski et al. 2012]. Many academic works on DVFS rely on fine-grained, per-core voltage regulation, but commercial processors continue to use coarse-grained, off-chip regulators. Given the uncertainty regarding DVFS, we leave determining the eventual state of DVFS and applying the LUCIE algorithm to DVFS for future work.

4.3. Candidate Set

The selection algorithms select cores from a candidate set (see figures 1 and 2). The example candidate set is generated as follows: We randomly sample 3000 configurations from the example design space and simulate all SPEC and EEMBC benchmarks on each configuration. Most of the simulated cores are not Pareto-optimal, and it would be computationally more efficient to implement an intelligent DSE algorithm to reduce the number of sub-optimal core simulated. However, random sampling allows us to avoid any hidden biases in DSE algorithms, and since DSE and selection are independent design stages, reducing the time spent on DSE does not affect the selection algorithms. Of the 3000 cores, 266 are power-performance Pareto-optimal for SPEC benchmarks, and 363 are optimal for EEMBC benchmarks. These Pareto-optimal sets are used as the candidate sets for the respective algorithms.

Since 3000 cores is a small sample size, we use the random search early stopping criterion proposed by Vuduc et al. [2004] to confirm that the sample is representative. For most benchmarks, there is $\geq$95% confidence that the best power and best performance found by random sampling is within 2% of the best possible in the design space. Exceptions are the performance results of the *aes*, *gcc*, *perlbench*, and *sjeng* benchmarks, where there is a $\geq$95% confidence that the best performance in the sample is within

5% of the best possible in the design space. We conclude that the sample of 3000 cores is sufficiently representative of the entire design space.

5. EVALUATION METHODOLOGY

A heterogeneous processor in a mobile device must provide a range of operating points to maximize performance under various power budgets. In the following, we first provide details on our implementations of the LUCIE and Clustering algorithms. We then describe the two metrics we use to compare the algorithms: speedup and effective speed. Common evaluation metrics, like IPC (instructions per cycle), ED^2 (energy-delay-squared product) [Martin et al. 2002], and STP (system throughput) [Eyerman and Eeckhout 2008] can only be used with fixed power budgets. As noted in section 2.1, the power available to a program on a mobile device varies over time. We use a speedup metric and an effective speed metric that evaluate the performance of sets of heterogeneous cores under probabilistically varying power budgets.

5.1. Algorithm Implementations

We implement the LUCIE algorithm as described in section 3 using a perl script. We implement the Clustering selection algorithm in R [R Core Team 2013] using the default R implementation of k-means clustering. Cores are clustered based on the $BIPS^3/W$ efficiency metric, and the representative core from each cluster is selected to minimize the coefficient of variation (CoV) of $BIPS^3/W$. Using $BIPS^3/W$ for clustering and selection has been found to lead to the best results [Guevara et al. 2014]. We note that results would be very similar if using ED or ED^2 instead of $BIPS^3/W$, as the three metrics are directly correlated.

Both the LUCIE algorithm and the Clustering algorithm require only a few seconds to select cores from a candidate set. The majority of time taken by the design flow in figure 1 is spent searching for a candidate set. For example, simulating our dataset of 33 benchmarks and 3000 cores required approximately 16 compute-years, though a better DSE algorithm could substantially reduce the required simulation time (see section 4.3). Since both selection algorithms require a candidate set, the search time is orthogonal to the time required to select cores. We note, however, that the Clustering selection is based on k-means, which is a non-deterministic algorithm. We found that k-means would sometimes fail to find good clusters of cores, and as a result, the selected cores would not implement a range of power-performance points. LUCIE is deterministic, and does not have this problem.

5.2. Speedup Metric

To measure speedup, we use the *set overhead* metric defined in our earlier work [Tomusk et al. 2015b]. Set overhead compares two selections of cores, and evaluates how much slower the slower one is in the average case. For example, if selection B has a set overhead of 20% compared to selection A for a given benchmark, then the benchmark will require (on average) 20% more time on B than on A. It can be said that B has a 20% slowdown compared to A, or A provides a 20% speedup over B (since speedup is defined as new execution time divided by old execution time). Rather than measure speedup for a fixed power budget, set overhead evaluates speedup across all possible power budget (because on a mobile device, the amount of power available to a task changes over time). I.e., set overhead uses a probabilistically determined available power budget. Set overhead has been designed to be robust to absolute errors in simulators and power models, and only requires that cores' power and performance be accurate relative to other cores. In our main evaluations in section 6, we assume that the probability density function (PDF) of available power is flat. We demonstrate the

use of an empirically determined non-flat power PDF in section 8. We also assume that the maximum amount of power ever available to a benchmark (the upper-bound of the PDF) is 10% greater than the maximum power consumed by the most power-hungry core in either selection when executing the benchmark.

5.3. Effective Speed Metric

The *effective speed* metric (ES) measures how fast a set of heterogeneous cores is compared to the fastest possible core when the fastest core has access to unlimited power. Like speedup above, ES uses a probabilistically varying power budget. ES accounts for the time a program is stalled because there is insufficient power to use any core. For example, if a heterogeneous processor can always use the fastest core, then ES is 1.0. If a program uses the fastest core half of the time, and is stalled the other half because power is limited, then ES is 0.5. Effective speed was proposed but not defined in our earlier work [Tomusk et al. 2015b].

We define effective speed with equation 6. ES_b is the effective speed of the selection of cores for benchmark b. $\overline{T_b}$ is the average execution time of b on the selection of cores, normalized to the fastest core in the design space. It is an average of the normalized execution time of b on each selected core, weighted by the likelihood that the core will be used. This likelihood is the likelihood that there is enough power to use the core, but not enough power to use a faster core. Av_b is the *availability* of the selection—the likelihood that b can be run at all. N is the number of selected cores. P_i and T_i are the power and execution time of core i for b, normalized as described in section 3.1.1. Cores are ordered from low to high power. P_{max} is the maximum power ever available to b. As with effective speed, we set P_{max} to 10% greater than the maximum power of b on any core in the comparison, and except for section 8, we assume a flat probability distribution for power.

$$ES_b = \left(\overline{T_b} \times \frac{1}{Av_b}\right)^{-1}$$

$$\overline{T_b} = \frac{T_N \times (P_{max} - P_N) + \sum_{i=1}^{N-1} T_i \times (P_{i+1} - P_i)}{P_{max} - P_1}$$

$$Av_b = \frac{P_{max} - P_1}{P_{max} - 1} \quad (6)$$

6. RESULTS

We evaluate the LUCIE selection algorithm and the Clustering selection algorithm on the SPECint 2006 and EEMBC benchmark suites. We use both algorithms to select four cores for SPEC and four cores for EEMBC. Along with previous work, we have found that there are diminishing returns to using more than four different types of cores [Navada et al. 2013]. As the number of cores increases, each core makes a smaller contribution to the diversity of the selection.

We find that the risk minimization approach taken by the Clustering algorithm [Guevara et al. 2014] leads to unnecessarily extreme cores, while the Pareto frontier optimization approach of LUCIE selects more moderate cores that are appropriate for mobile devices. This section will discuss the cores selected by both algorithms. We evaluate the speedup provided by the LUCIE algorithm over the Clustering algorithm, and we compare the effective speeds of the cores selected by the algorithms. We consider the *scalability* of the algorithms—how the selected cores compare when more than four types are selected. Finally, we use LUCIE to tune a selection of cores to only a subset of a benchmark suite.

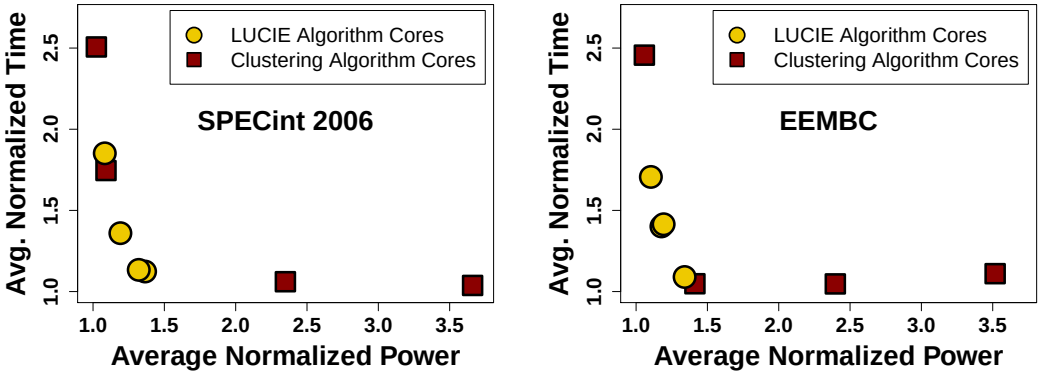

Fig. 4. Four cores selected by LUCIE and the Clustering algorithm for the SPECint 2006 suite (left) and the EEMBC suite (right). Configurations for the SPECint selections are shown in tables IV and V.

6.1. Selected Cores

Figure 4 shows four cores selected by LUCIE and the Clustering algorithm for both the SPECint and EEMBC suites. As noted in section 3.2 and shown in figure 3 (bottom-right), averages hide per-benchmark behavior variations. This causes some cores to appear closer to each other in the averaged power-performance space than they actually are in the space of any one benchmark. The microarchitectural parameters for the four SPEC cores selected by LUCIE are listed in table IV; the parameters for the four SPEC cores selected by the Clustering algorithm are listed in table V. We have omitted the equivalent tables for the EEMBC suite due to space considerations. While both selection algorithms are agnostic to the implementation details of the cores (see section 4.2), analyzing the types of cores selected by each algorithm is helpful for understanding the algorithms' behaviors.

In our design space, power and performance are primarily controlled by the data cache and the integer register file (*IR*). Both of these structures are key to extracting instruction level parallelism (ILP), but their size and complexity also makes them significant consumers of power. At larger sizes, these structures become particularly expensive due to the additional complexity required to meet timing. The instruction cache is not nearly as significant as the data cache, since the i-cache has only one hardware port, compared to the d-cache's two. Floating point registers are not as significant as integer registers, because our benchmarks perform few floating point operations. After the d-cache and *IR*, the queues also have a significant effect on power and performance. While queues consume little power themselves, they enable greater ILP and greater power consumption throughout the core. We find that the branch predictor has little to no impact on benchmark performance. This has also been noted elsewhere [Sunwoo et al. 2013], and may be due to the gem5 implementation of branching logic.

The behavioral differences between LUCIE and the Clustering selection algorithms are most obvious in the selected cores' data cache sizes (*DS*) and integer register file sizes (*IR*) in tables IV and V. Recall that our LUCIE algorithm approximates the power-performance Pareto-optimal frontier for all benchmarks using just a few cores. In contrast, the Clustering algorithm clusters cores by efficiency, and then selects from each cluster the core that is most consistent (has the lowest coefficient of variation, CoV) for efficiency across all benchmarks. The crucial limitation of the Clustering approach is that maximizing consistency is equivalent to taking the lowest common denominator—there is no scope for taking advantage of behavioral differences among

Table IV. Configurations for the four cores (1-4) selected by LUCIE for SPEC benchmarks (see figures 3 and 4 (left)). Parameter names are in table III. Cores are ordered from low power to high performance. Cores 5 and 6 are *pinned cores* discussed in section 7.

	Caches				Registers		Queues				Branch Predictor							BTB	
Core	**DS**	**DW**	**IS**	**IW**	**IR**	**FPR**	**IQ**	**LQ**	**SQ**	**ROB**	**GCB**	**GE**	**LCB**	**LHB**	**LHE**	**CCB**	**CE**	**BE**	**BT**
1	16kB	1	8kB	1	96	128	16	16	16	16	2	2^{14}	2	12	2^{10}	2	2^{12}	2^{12}	18
2	16kB	2	32kB	4	96	128	16	64	16	128	3	2^{14}	1	11	2^{9}	1	2^{14}	2^{10}	20
3	32kB	2	64kB	4	96	256	32	32	64	40	3	2^{14}	2	11	2^{11}	2	2^{12}	2^{13}	16
4	32kB	2	32kB	4	128	96	64	32	32	128	2	2^{13}	1	12	2^{10}	2	2^{13}	2^{10}	20
***5**	32kB	1	4kB	1	64	96	16	64	8	16	3	2^{12}	1	10	2^{11}	3	2^{14}	2^{11}	18
***6**	64kB	2	64kB	2	128	256	32	64	32	64	2	2^{14}	2	12	2^{11}	2	2^{10}	2^{13}	18

Table V. Configurations for the four cores selected by the Clustering algorithm for SPEC benchmarks (see figure 4 (left)). Parameter names are in table III. Cores are ordered from low power to high performance.

	Caches				Registers		Queues				Branch Predictor							BTB	
Core	**DS**	**DW**	**IS**	**IW**	**IR**	**FPR**	**IQ**	**LQ**	**SQ**	**ROB**	**GCB**	**GE**	**LCB**	**LHB**	**LHE**	**CCB**	**CE**	**BE**	**BT**
1	16kB	2	64kB	4	50	128	16	16	16	32	1	2^{14}	1	12	2^{11}	3	2^{13}	2^{10}	18
2	16kB	1	64kB	1	64	128	16	32	32	16	1	2^{10}	2	10	2^{11}	2	2^{14}	2^{11}	20
3	32kB	1	64kB	2	256	256	64	64	32	128	3	2^{13}	3	11	2^{10}	3	2^{14}	2^{11}	18
4	64kB	1	32kB	4	256	256	64	64	32	128	3	2^{14}	1	10	2^{11}	2	2^{13}	2^{12}	20

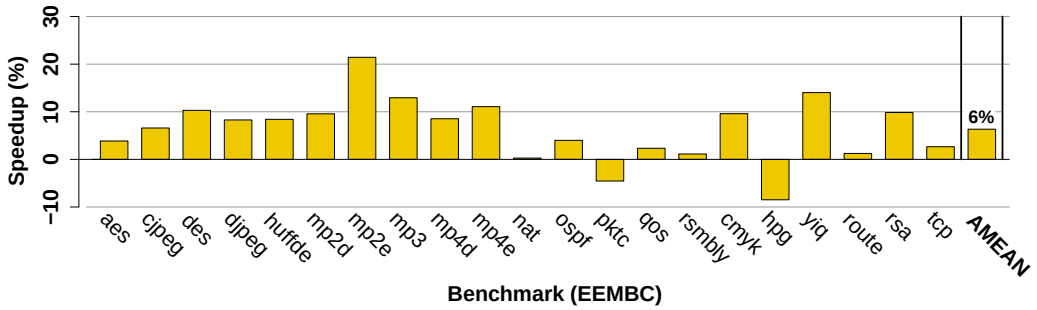

Fig. 5. Speedup of using cores selected by LUCIE instead of cores selected by the Clustering algorithm for the EEMBC suite. The speedup metric is based on execution time and requires the arithmetic mean.

benchmarks if cores must have similar efficiency for all benchmarks. For example, the slowest core selected by the Clustering method has 50 integer register file entries, whereas none of the cores selected by LUCIE have fewer than 96. For the register file, the power difference between 50 and 96 entries is insignificant. The Clustering method uses a core with 50 *IR* entries because this is an artificial bottleneck that keeps all benchmarks at a certain power and performance level. A larger *IR* would accelerate some benchmarks more than others, thereby decreasing consistency. Similar behavior can be observed with the largest core selected by the Clustering method, which contains both a 64kB d-cache and a 256-entry *IR*. None of the cores selected by LUCIE contain a data cache or integer register file this large. These over-sized structures ensure that this largest core consistently consumes substantial power for all benchmarks, even though some benchmarks do not benefit from such large structures.

6.2. Speedup

We measure speedup as described in section 5.2. The speedup metric is based on execution time, not execution rate, and similarly to the $ANTT$ metric [Eyerman and Eeckhout 2008], it must be summarized with the arithmetic mean.

Figure 5 shows the speedup of using cores selected by LUCIE instead of cores selected by the Clustering algorithm for the EEMBC suite. On average, LUCIE provides a 6% speedup over the Clustering algorithm, reaching 21% for *mpeg2enc*. The Clustering selection is, however, faster than the LUCIE selection for *ip_pktcheck* and

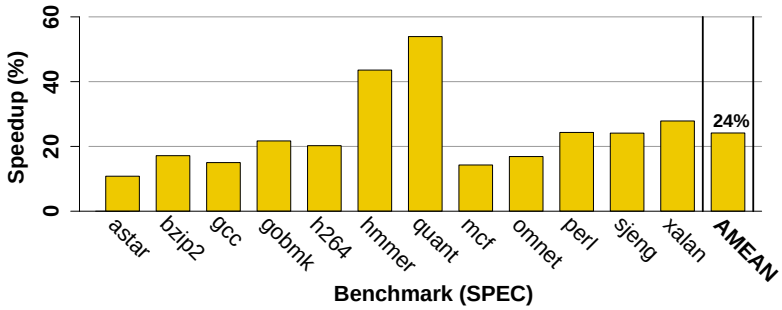

Fig. 6. Speedup of using cores selected by LUCIE instead of cores selected by the Clustering algorithm for the SPEC suite. The speedup metric is based on execution time and requires the arithmetic mean.

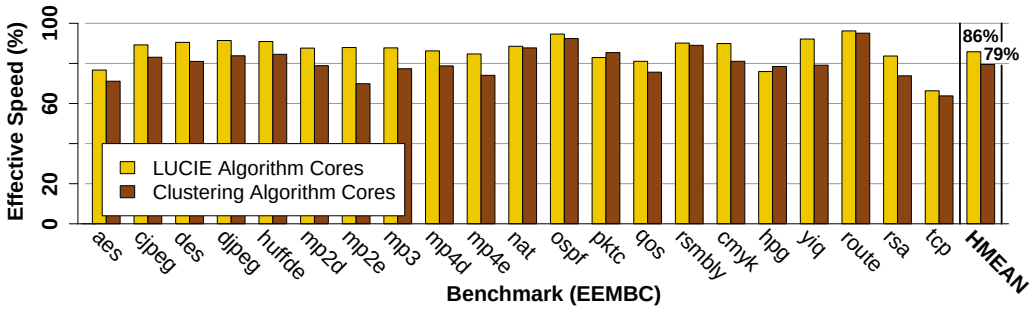

Fig. 7. Cores selected by LUCIE provide, on average, 7pp more speed for the EEMBC suite than cores selected by the Clustering algorithm. The Clustering algorithm provides a speed advantage to only two of the 21 EEMBC benchmarks.

rgbhpgv2 by 5% and 8% respectively. These two benchmarks are able to take advantage of the larger microarchitectural structures selected by the Clustering algorithm (see section 6.1 above). In most cases, slowing down two benchmarks to speed up 19 is an acceptable trade-off. In section 7, we will consider the situation where maximum performance is required. Figure 6 shows the same comparison for the SPEC suite. LUCIE provides an average 24% speedup to SPEC benchmarks. Performance is most similar for the *astar* benchmark, where speedup is only 11%. On *libquantum*, speedup from using LUCIE is 54%.

6.3. Effective Speed

Effective speed, ES, measures how close the average speed of a benchmark on a set of heterogeneous cores comes to maximum possible speed (section 5.3). ES is a measure of rate, and is summarized with the harmonic mean.

Figure 7 shows the effective speed of the EEMBC benchmarks on both the LUCIE and the Clustering selections of cores. As already seen in the speedup comparison, LUCIE provides greater speed to all but two of the EEMBC benchmarks. On average, ES is 7pp (percentage points) greater with the LUCIE selection than with the Clustering selection. Figure 8 shows the ES comparison for SPEC benchmarks. On average, speed on the LUCIE selection is 16pp greater than speed on the Clustering selection. This is less than the reported speedup (figure 6), because ES accounts for the time when there is so little power available that no core can be used. The lowest-power core in the Clustering selection has slightly lower power than the lowest-power core in the

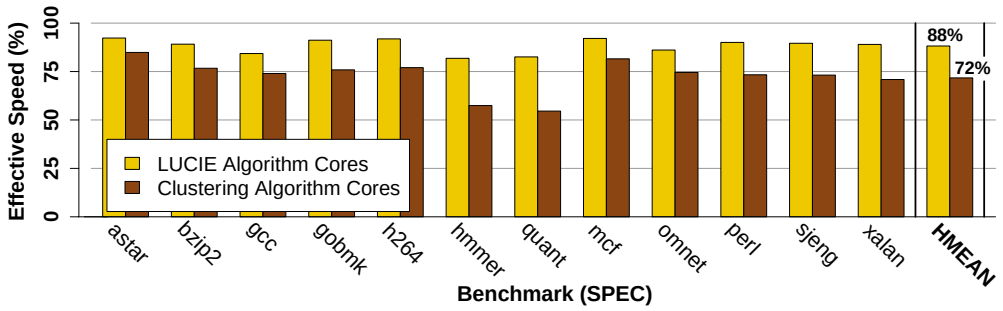

Fig. 8. Cores selected by LUCIE provide, on average, 16pp more speed for the SPEC suite than cores selected by the Clustering algorithm.

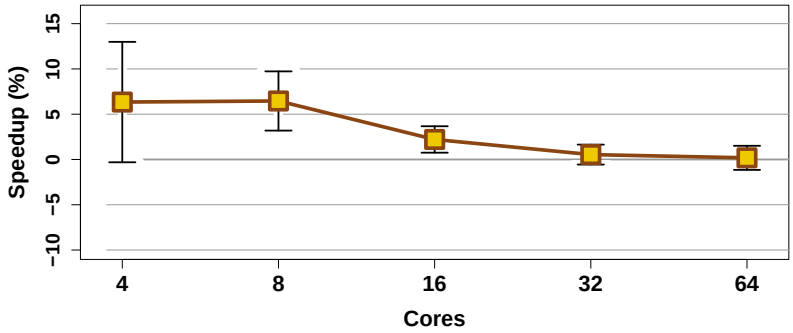

Fig. 9. Mean speedup of using cores selected by LUCIE instead of cores selected with the Clustering method for the EEMBC suite. Error bars are standard deviations.

LUCIE selection, allowing the Clustering selection to make progress in some corner cases where the LUCIE selection cannot.

6.4. Scalability

We now consider the *scalability* of the selection algorithms—how the selected cores compare as the number of selected cores changes. Figure 9 shows the speedup (see section 5.2) of the LUCIE selection over the Clustering selection from four core types up to 64 types. In the majority of cases, the LUCIE algorithm selects a faster set of cores than the Clustering algorithm. For four and eight cores, the LUCIE selection is, on average, over 6% faster.

As the number of cores increases, both selection algorithms converge on selecting the entire candidate set, and the difference between the algorithms becomes insignificant. In our example space, LUCIE provides the maximum speedup over the Clustering algorithm at eight core types. At 64 types, the LUCIE selection is less than 1% faster than the Clustering selection. The point where the selections converge is dependent on the design space and the size of the candidate set.

6.5. Selection Specialization

The set of benchmarks used to tune a processor during the design process has far-reaching effects on the processor. If the wrong set of benchmarks is chosen, then the processor loses out on potential performance and power improvements. Conversely, if the processor designer can correctly narrow the scope of the benchmarks used to select cores, then further performance improvements are possible. We will now show that LUCIE can specialize a selection of cores to deliver this additional performance.

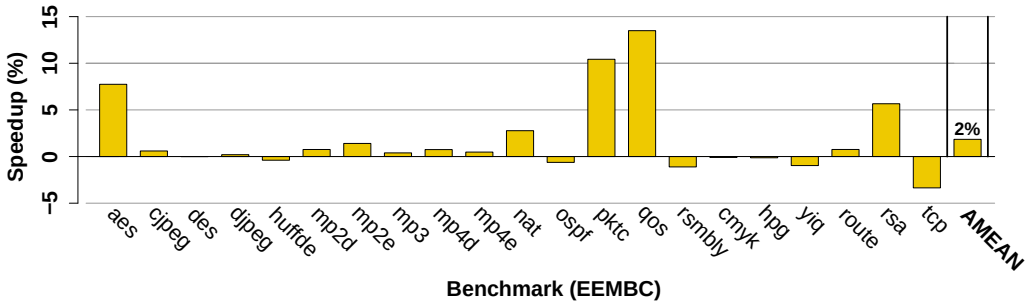

Fig. 10. Speedup when running EEMBC benchmarks on sets of cores selected for subsets of the suite rather than the entire suite. EEMBC subsets are shown in table II.

We divide the EEMBC benchmark suite into three classes: cryptography, audio-video, and networking (see table II). We then use LUCIE to select four cores for each class, and measure the speedup of using the cores selected for a benchmark's class rather than the cores selected for the entire EEMBC suite. That is, we compare a general, EEMBC processor to processors designed for cryptography, audio-video, and networking. For example, the *aes* benchmark receives an 8% speedup when run on cores selected for cryptography instead of the set of general EEMBC cores. The results are summarized in figure 10. Average speedup from specialization is 2%.

While specialization can improve performance, over-specialization can hurt performance. For example, *tcp* is an outlier among the networking benchmarks, so while specialization provides an average improvement of 3% to the networking benchmarks, *tcp* is slowed down by 3%. Depending on the priority of *tcp*, this may or may not be acceptable to the designer. Similarly, specializing a selection to the wrong benchmarks also leads to a slowdown. Running the audio-video benchmarks on the networking set of cores incurs an average slowdown of 5%, for example.

This result shows that LUCIE can extract additional performance from a processor if the processor will be used for a narrow set of programs, but care must be taken to identify the correct set of programs to which the processor will be tuned.

7. CORE PINNING

By default, the LUCIE algorithm eliminates cores that are Pareto-optimal for only a few benchmarks, and cores that trade disproportionately large amounts of power for marginal performance improvements (or vice-versa). The underlying assumption is that these cores are a waste of silicon resources—cores that are optimal for only a few benchmarks will only rarely be required, and cores that consume large amounts of power will only rarely be usable in a power-limited device. In some cases, however, a designer might have good reason to force LUCIE to select a core that would normally be excluded. We refer to this as *core pinning*. A core, c is pinned by setting its cost, $\underline{\mathbf{C}}(c)$ (equation 2), to infinity. LUCIE is run normally, benchmark affinities are calculated even for the pinned cores, but the pinned cores are never removed. We provide two use cases for pinning: maximizing best-case execution speed and incrementally designing processors. Pinning is completely optional, and a designer can choose not to pin any cores. We note that pinning is a unique feature of LUCIE—it is not clear how pinning could be implemented for the Clustering algorithm.

7.1. Performance Maximization

LUCIE assumes that the power available to a program can vary, but some types of programs might be so important that the device will always make enough power avail-

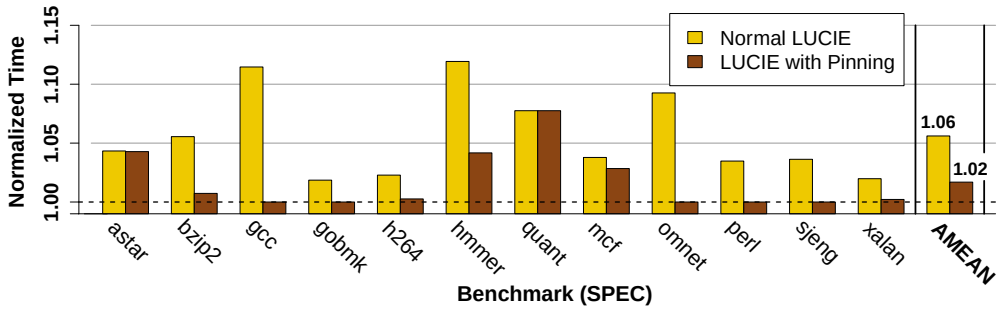

Fig. 11. Pinning a fast core increases speed for most benchmarks when power is not limited (smaller is better).

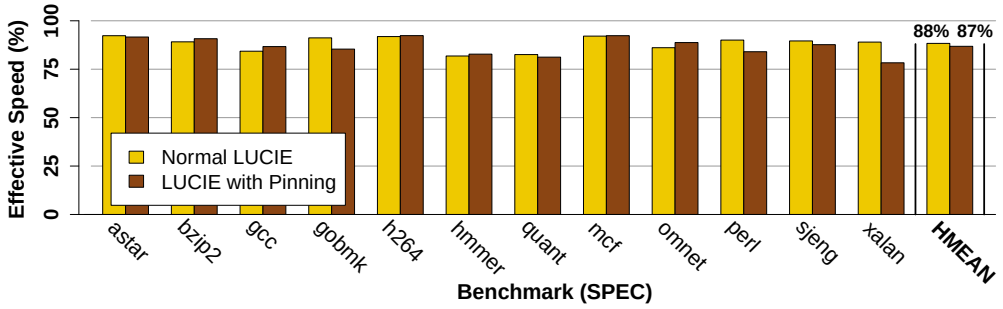

Fig. 12. Pinning a fast core slightly reduces the effective speed of benchmarks (larger is better).

able to run them as fast as possible (by, e.g., pausing other running programs, turning off radio interfaces, dimming the screen, etc.). For such a device, the designer may choose to pin a fast, high-power core. We demonstrate this for the SPEC benchmarks by pinning core #6 in table IV. Core #6 is chosen because of its large caches and integer register file, and because it is Pareto-optimal for half of the SPEC benchmarks (most fast cores are optimal for fewer benchmarks). When core #6 pinned, LUCIE selects cores #1, #2, and #4, but not core #3. Depending on the benchmarks and design space, LUCIE might select completely different cores to complement a pinned core. Figure 11 shows best-case execution time for SPEC benchmarks when power is not limited. Pinning improves best-case execution time by an average of nearly 4%. The pinned core is not able to run *libquantum* faster than any of the cores normally selected by LUCIE.

Pinning is not, however, an unqualified improvement, but a trade-off between best-case and average performance. Pinning a fast core can reduce overall effective speed, because the fast core requires more power and can be used less frequently. Figure 12 shows that for this example, pinning causes only a 1% reduction in ES, as the selection of cores is not substantially different. The side effects of pinning will be more drastic if a core with a very low affinity is pinned, or if more core types are pinned.

7.2. Incremental Design

A second use of core pinning is if the designer has already implemented some cores (for, e.g., a previous product), and wishes to select additional cores to complement the existing ones. In this case, the designer simply pins the existing cores and runs LUCIE as normal. We demonstrate with figure 13, where we have pinned two cores: #5 and #6 from table IV. The performance difference between the pinned cores is approximately $2.0\times$, which is similar to the performance difference between core types

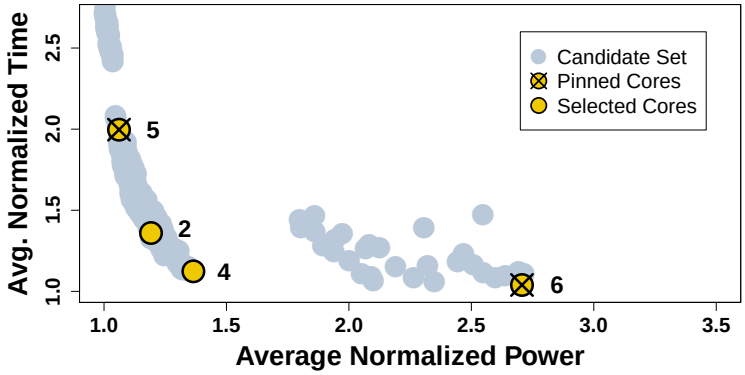

Fig. 13. When a slow and a fast core are pinned, LUCIE selects cores in the middle. Core configurations are in table IV.

in ARM's *big.LITTLE* configuration [Greenhalgh 2011]. With cores #5 and #6 pinned, LUCIE selects core #4 as the third core. Adding core #4 leads to a speedup of 56% compared to the 2-core case. If LUCIE is allowed to select a fourth core, core #2 is selected. This increases the speedup to 61% over the 2-core case.

8. APPLYING A POWER PDF

In some circumstances, a designer might wish to guide the selection process without pinning specific cores. For example, the designer might know that real usage patterns emphasize a specific region of the design space. By using an available power PDF (probability density function), it is possible to increase the cost of cores in important regions of the space. LUCIE still selects cores for power flexibility and to balance the needs of all benchmarks, but it places a greater emphasis on the important regions. Like pinning, the use of a power PDF is unique to the LUCIE algorithm; a PDF cannot be used to affect the cores selected by the Clustering algorithm. We describe how the LUCIE algorithm can be made aware of a non-flat power PDF and provide an example.

8.1. Usage

The available power PDF is used as follows: The PDF applies a factor, $f(c, b)$, to the displacement cost function, $\underline{\mathbf{D}}(c, b)$, as shown in equation 7. The PDF modifies the cost of displacing core c for each benchmark b. $f(c_i, b)$ is defined in equation 8. It is the integral of the PDF for benchmark b from the midpoint between core c_i and its neighbor on the left to the midpoint between c_i and its neighbor to the right. Cores are ordered by increasing power. The integration range for the first and last core begins at the lower-bound of the PDF and ends at the upper-bound of the PDF, respectively. Integrating with respect to a core's neighbors ensures that the PDF in a given region of the design space is divided among the cores in that region. As cores are removed, the remaining cores gain an increased benefit from the PDF. This prevents tight clusters of cores forming in regions of the design space with a high probability density. The PDF is defined on a per-benchmark basis. Low-priority tasks and tasks that are threads of multithreaded programs might always have small amounts of power available. High-priority tasks might often have large amounts of power available.

$$\begin{aligned} \underline{\mathbf{D}}(c, b) &= f(c, b) \times \underline{\mathbf{BD}}(c, cm, b) \\ cm &= \arg\min_{k} \ \underline{\mathbf{BD}}(c, k, b) \end{aligned} \tag{7}$$

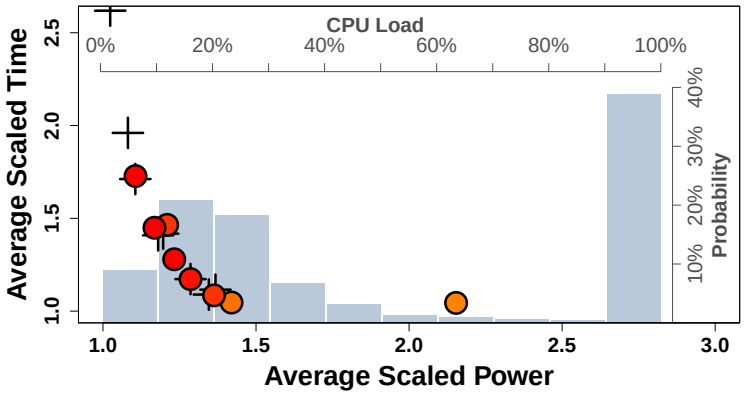

Fig. 14. With this particular power PDF, LUCIE selects more high-power cores. The highest-power core has a lower affinity. Cores selected without using the PDF are indicated by a + for reference.

$$
\begin{aligned}
f(c_i, b) &= \int_{l_1}^{l_2} \mathrm{PDF}_b \\
l_1 &= \frac{\mathrm{P}_{\mathrm{raw}}(c_{i-1}, b) + \mathrm{P}_{\mathrm{raw}}(c_i, b)}{2} \\
l_2 &= \frac{\mathrm{P}_{\mathrm{raw}}(c_i, b) + \mathrm{P}_{\mathrm{raw}}(c_{i+1}, b)}{2}
\end{aligned}
\tag{8}
$$

8.2. Example

The application of a power PDF is demonstrated in figure 14. The plot shows an empirical distribution of CPU load data collected from a modern Android smartphone. I.e., it shows how likely the phone is to experience a given load. For this example, we simplistically assume that the amount of power available to the CPU correlates with the load—if the load is low, it is because there is insufficient power to run a greater load. In reality, determining the source of CPU load is far more involved. However, the load distribution is a reasonable example of the amount of power that a scheduler may make available to a task, and is sufficient for illustrating the application of a power PDF. The PDF is scaled to the power axis using the range of power values in the candidate set. The smartphone CPU spends much of its time at 0%-30% load, and also at 90%-100% load. Given the above assumption, this informs LUCIE that cores that can operate at below 30% of full power and at 90% of full power are more useful than other cores. For this example, the same PDF is used with all benchmarks.

The effects of the PDF on a selection are easiest to see when eight cores are selected. Figure 14 shows the LUCIE selection for the EEMBC suite given the PDF. The eight cores selected for EEMBC when no PDF is given are indicated with + signs. It can be seen that with the PDF, LUCIE selects more cores in the 10%-30% region, and one high-power (but low-affinity) core. We note that the PDF only steers LUCIE—it does not completely dominate. The trade-off mechanisms in LUCIE still apply, and only one core with disproportionately high power is introduced due to the PDF.

The power PDF can similarly be applied to the metrics to measure speedup and effective speed under a non-flat distribution of available power. Figure 15 shows the effective speeds of selections under the power PDF from figure 14. Under this PDF, the set of four cores selected by LUCIE has an ES of 78%. When LUCIE is aware of the PDF, ES increases to 80%. The Clustering algorithm does not have a mechanism

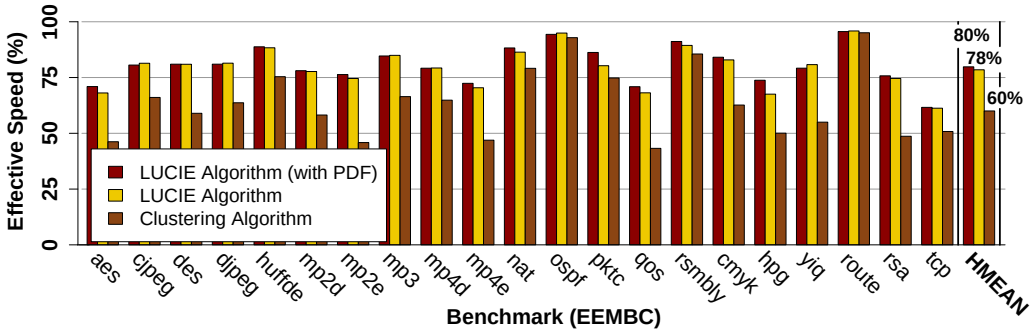

Fig. 15. When evaluating ES under a non-flat power PDF, LUCIE can improve performance if it is aware of the PDF. There is no way to communicate a power PDF to the Clustering algorithm, causing its performance to suffer. Each selection contains four cores.

for selecting cores using a PDF. With this particular PDF, the effective speed of the Clustering algorithm drops to only 60%

9. RELATED WORK

There is a large body of research on design space exploration (DSE). Some authors build analytical models of design spaces [Lee and Brooks 2006; Lee and Brooks 2007; Karkhanis and Smith 2007]. Others develop methods to quickly find optimal cores [Kang and Kumar 2008; Azizi et al. 2010; Liu et al. 2011; Sunwoo et al. 2013; Turakhia et al. 2013] and models to predict the optimal performance on cores [Dubach et al. 2008]. The goal for many of these studies is to characterize the design space and to find a comprehensive set of Pareto-optimal cores. Core selection algorithms are required for choosing which of the cores found by DSE should be implemented [Tomusk et al. 2015a]. Kumar et al. [2006] select sets of heterogeneous cores to maximize performance. Navada et al. [2013] perform DSE and core selection at once using a genetic algorithm that also optimizes for performance. The current state of the art in core selection clusters cores, and selects a representative core from each cluster [Guevara et al. 2014].

Instead of selecting cores from a large design space, some works compose processors from a small number of extant cores [Kumar et al. 2003; Annavaram et al. 2005; Greenhalgh 2011; Van Craeynest and Eeckhout 2013]. However, as noted by Kumar et al. [2006], a core that has been designed for a homogeneous processor is not necessarily a good choice for a heterogeneous processor, as heterogeneity allows cores to be optimized for one type of program or one power-performance point.

There is some disagreement regarding the role of heterogeneity in servers. The Clustering selection method chooses cores to minimize runtime risk for server workloads [Guevara et al. 2014]. Ren et al. [2014] argue that a broad spread of heterogeneous cores is needed to minimize energy while meeting quality-of-service requirements. Others have found that it is sufficient for a server to have two types of cores [Van Craeynest and Eeckhout 2013], or even just one type of core with SMT [Eyerman and Eeckhout 2014]. However, the latter two works do not consider issues like risk, or the variable power budgets encountered by mobile devices.

A number of studies have extended heterogeneity to include cores that have specialized instruction set extensions [Venkatesh et al. 2010; Goulding-Hotta et al. 2011; Venkatesh et al. 2011], or even cores with completely different ISAs [DeVuyst et al. 2012]. Scheduling [Li et al. 2010] and compilation for such processors is the topic of

ongoing research. Since LUCIE does not consider the implementation details of cores, it can easily be extended to select cores from a multiple-ISA space.

For our analysis, we have assumed an oracle scheduler. Energy- and performance-aware schedulers for heterogeneous and DVFS-enabled processors are presented by Zhu and Reddi [2013] and Lukefahr et al. [2014]. A learning phase is used by Alsafrjalani and Gordon-Ross [2014] to determine which heterogeneous core a task should be scheduled to. Su et al. [2014] predict power and performance for a DVFS-enabled homogeneous processor, but the framework could be extended to heterogeneous processors. Panneerselvam and Swift [2016] implement a power-allocating scheduler similar to the one LUCIE requires.

There is ongoing debate regarding the relationship between power consumption and energy consumption. Some authors have found that energy is minimized when performance and power are maximized ("sprinting"), while others have found that minimizing performance and power also minimizes energy ("pacing"). The issue is addressed by Le Sueur and Heiser [2011] and Efraim et al. [2014]. Choi et al. [2004], Dhiman et al. [2008], and Raghavan et al. [2013] report that sprinting is preferable. Zhu and Reddi [2013] and Lukefahr et al. [2014] report that pacing is preferable. We find that our design space supports sprinting, and we therefore assume a scheduler that attempts to maximize performance under power constraints. When LUCIE is used with a design space that exhibits pacing behavior, the runtime scheduler must consider the trade-off between quality of service and energy consumption, in addition to considering power consumption. This may mean, for example, that the scheduler will regularly make small amounts of power available to tasks even when large amounts of power are available to the processor.

A number of methods have been suggested for evaluating multicore processors, including heterogeneous processors [Snavely and Tullsen 2000; Eyerman and Eeckhout 2008; Eyerman et al. 2014]. These works focus on server-type systems where throughput is a critical design requirement. The LUCIE algorithm targets power-limited, mobile devices. As a result, we have opted to use the set overhead and effective speed metrics [Tomusk et al. 2015b], since these metrics account for variability in the amount of power available at runtime.

LUCIE selects cores by preserving a Pareto-optimal frontier. Sampling a Pareto frontier is a common problem for genetic algorithms (GAs) [Zitzler and Thiele 1999; Deb et al. 2002]. However, the GA solutions do not directly apply to core selection, as GAs select from one Pareto frontier, whereas LUCIE selects cores to satisfy many Pareto frontiers (one for each benchmark).

10. CONCLUSION & FUTURE WORK

There has been considerable research effort in design space exploration (DSE) for heterogeneous processors. The problem of selecting which of the cores found by DSE should be implemented has, however, been largely unaddressed. Existing works on core selection target server processors. These works are not appropriate for mobile devices, as mobile processors must execute a range of different types of programs at a range of power-performance points. We have presented the LUCIE algorithm for iteratively reducing the size of a candidate set of heterogeneous cores down to the number of required cores, while preserving the power-performance trade-off in the original candidate set. LUCIE is based on the observation that a core does not need to have consistent behavior for different types of programs to be useful to different programs. Each core is evaluated based on its contribution to each benchmark. This insight leads to an average speedup of 6% for EEMBC benchmarks, and 24% for SPECint benchmarks, compared to the state of the art selection technique.

We have demonstrated LUCIE on a design space of out-of-order CPU cores. Future work includes extending this to larger design spaces that contain in-order cores and potentially more exotic architectures. Since LUCIE is oblivious to implementation details, it should be readily applicable to design spaces containing, e.g., specialized accelerators. As the future of DVFS becomes clearer, LUCIE could also be extended to select both DVFS levels and core types. This would require a more detailed cost model, since LUCIE would need to determine the relative benefits of adding a DVFS level or a core to a processor.

ACKNOWLEDGMENTS

We thank Peter Henderson for providing access to his collected CPU load data. This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF). (http://www.ecdf.ed.ac.uk/). The ECDF is partially supported by the eDIKT initiative (http://www.edikt.org.uk).

REFERENCES

Mohamad Hammam Alsafrjalani and Ann Gordon-Ross. 2014. Dynamic Scheduling for Reduced Energy in Configuration-Subsetted Heterogeneous Multicore Systems. In *International Conference on Embedded and Ubiquitous Computing (EUC)*. http://dx.doi.org/10.1109/EUC.2014.12

Murali Annavaram, Ed Grochowski, and John Shen. 2005. Mitigating Amdahl's Law through EPI throttling. In *International Symposium on Computer Architecture (ISCA)*. DOI:http://dx.doi.org/10.1109/ISCA.2005.36

Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. 2010. Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis. In *International Symposium on Computer Architecture (ISCA)*. DOI:http://dx.doi.org/10.1145/1815961.1815967

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (Aug. 2011), 7. DOI:http://dx.doi.org/10.1145/2024716.2024718

Kihwan Choi, Wonbok Lee, Ramakrishna Soma, and Massoud Pedram. 2004. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *International Conference on Computer Aided Design (ICCAD)*. DOI:http://dx.doi.org/10.1109/ICCAD.2004.1382538

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (Apr. 2002). DOI:http://dx.doi.org/10.1109/4235.996017

Matthew DeVuyst, Ashish Venkat, and Dean M. Tullsen. 2012. Execution migration in a heterogeneous-ISA chip multiprocessor. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. DOI:http://dx.doi.org/10.1145/2150976.2151004

Gaurav Dhiman, Kishore Kumar Pusukuri, and Tajana Rosing. 2008. Analysis of dynamic voltage scaling for system level energy management. In *USENIX Workshop on Power Aware Computing Systems (HotPower)*.

Christophe Dubach, Timothy M. Jones, and Michael F.P. O'Boyle. 2008. Exploring and Predicting the Architecture/Optimising Compiler Co-design Space. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. DOI:http://dx.doi.org/10.1145/1450095.1450103

Rotem Efraim, Ran Ginosar, Uri Weiser, and Avi Mendelson. 2014. Energy Aware Race to Halt: A Down to EARtH Approach for Platform Energy Management. *Computer Architecture Letters* 13 (Jan. 2014). DOI:http://dx.doi.org/10.1109/L-CA.2012.32

Maja Etinski, Julita Corbalán, Jesús Labarta, and Mateo Valero. 2012. Understanding the future of energy-performance trade-off via DVFS in HPC environments. *J. Parallel and Distrib. Comput.* 72, 4 (2012), 579–590. DOI:http://dx.doi.org/10.1016/j.jpdc.2012.01.006

Stijn Eyerman and Lieven Eeckhout. 2008. System-level performance metrics for multiprogram workloads. *IEEE Micro* 28, 3 (May. 2008), 42–53. DOI:http://dx.doi.org/10.1109/MM.2008.44

Stijn Eyerman and Lieven Eeckhout. 2014. The Benefit of SMT in the Multi-core Era: Flexibility Towards Degrees of Thread-level Parallelism. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. DOI:http://dx.doi.org/10.1145/2541940.2541954

Stijn Eyerman, Pierre Michaud, and Wouter Rogiest. 2014. Multiprogram throughput metrics: A systematic approach. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 3 (Oct. 2014). DOI:http://dx.doi.org/10.1145/2663346

Nathan Goulding-Hotta, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Po-Chao Huang, Manish Arora, Siddhartha Nath, Vikram Bhatt, Jonathan Babb, Steven Swanson, and Michael Bedford Taylor. 2011. The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future. *IEEE Micro* 31, 2 (Mar. 2011). DOI:http://dx.doi.org/10.1109/MM.2011.18

Peter Greenhalgh. 2011. *Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7*. White paper. ARM Ltd.

Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2014. Strategies for anticipating risk in heterogeneous system design. In *International Symposium on High Performance Computer Architecture (HPCA)*. DOI:http://dx.doi.org/10.1109/HPCA.2014.6835926

Sukhun Kang and Rakesh Kumar. 2008. Magellan: A Search and Machine Learning-based Framework for Fast Multi-core Design Space Exploration and Optimization. In *Design, Automation, and Test in Europe Conference (DATE)*. DOI:http://dx.doi.org/10.1145/1403375.1403721

Tejas S. Karkhanis and James E. Smith. 2007. Automated Design of Application Specific Superscalar Processors: An Analytical Approach. In *International Symposium on Computer Architecture (ISCA)*. DOI:http://dx.doi.org/10.1145/1250662.1250712

Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. 2003. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *International Symposium on Microarchitecture (MICRO)*. DOI:http://dx.doi.org/10.1109/MICRO.2003.1253185

Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. 2006. Core architecture optimization for heterogeneous chip multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 10. DOI:http://dx.doi.org/10.1145/1152154.1152162

Etienne Le Sueur and Gernot Heiser. 2010. Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns. In *Workshop on Power Aware Computing and Systems (HotPower)*.

Etienne Le Sueur and Gernot Heiser. 2011. Slow Down or Sleep, That is the Question. In *USENIX Annual Technical Conference (USENIXATC)*. http://dl.acm.org/citation.cfm?id=2002181.2002197

Benjamin C. Lee and David M. Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. DOI:http://dx.doi.org/10.1145/1168857.1168881

Benjamin C. Lee and David M. Brooks. 2007. Illustrative Design Space Studies with Microarchitectural Regression Models. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. http://dx.doi.org/10.1109/HPCA.2007.346211

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *International Symposium on Microarchitecture (MICRO)*. DOI:http://dx.doi.org/10.1145/1669112.1669172

Tong Li, Paul Brett, Rob Knauerhase, David Koufaty, Dheeraj Reddy, and Scott Hahn. 2010. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *International Symposium on High Performance Computer Architecture (HPCA)*. DOI:http://dx.doi.org/10.1109/HPCA.2010.5416660

Hung-Yi Liu, Ilias Diakonikolas, Michele Petracca, and Luca Carloni. 2011. Supervised Design Space Exploration by Compositional Approximation of Pareto Sets. In *Design Automation Conference (DAC)*. DOI:http://dx.doi.org/10.1145/2024724.2024818

Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Ronald Dreslinski Jr., Thomas F. Wenisch, and Scott Mahlke. 2014. Heterogeneous Microarchitectures Trump Voltage Scaling for Low-power Cores. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. DOI:http://dx.doi.org/10.1145/2628071.2628078

Alain J. Martin, Mika Nyström, and Paul I. Pénzes. 2002. ET^2: A metric for time and energy efficiency of computation. In *Power Aware Computing*, Robert Graybill and Rami Melhem (Eds.). Springer, 293–315. DOI:http://dx.doi.org/10.1007/978-1-4757-6217-4

Sandeep Navada, Niket K. Choudhary, Salil V. Wadhavkar, and Eric Rotenberg. 2013. A unified view of non-monotonic core selection and application steering in heterogeneous chip multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. http://dl.acm.org/citation.cfm?id=2523721.2523743

Sankaralingam Panneerselvam and Michael M. Swift. 2016. *Firestorm: Operating Systems for Power-Constrained Architectures*. Technical report. University of Wisconsin–Madison. http://digital.library.wisc.edu/1793/75140

Jason A. Poovey, Markus Levy, Shay Gal-On, and Thomas M. Conte. 2009. A Benchmark Characterization of the EEMBC Benchmark Suite. *IEEE Micro* 29, 5 (Sep. 2009), 18–29. DOI:http://dx.doi.org/10.1109/MM.2009.74

R Core Team. 2013. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/

Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. 2013. Computational sprinting on a hardware/software testbed. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 12. DOI:http://dx.doi.org/10.1145/2451116.2451135

Shaolei Ren, Yuxiong He, and Kathryn S. McKinley. 2014. A Theoretical Foundation for Scheduling and Designing Heterogeneous Processors for Interactive Applications. In *International Symposium on Distributed Computing (DISC)*. DOI:http://dx.doi.org/10.1007/978-3-662-45174-8_11

Allan Snavely and Dean M. Tullsen. 2000. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. DOI:http://dx.doi.org/10.1145/378993.379244

Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathouse, and Zhiying Wang. 2014. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *International Symposium on Microarchitecture (MICRO)*. DOI:http://dx.doi.org/10.1109/MICRO.2014.17

Dam Sunwoo, William Wang, Mrinmoy Ghosh, Chander Sudanthi, Geoffrey Blake, Christopher D. Emmons, and Nigel C. Paver. 2013. A structured approach to the simulation, analysis and characterization of smartphone applications. In *International Symposium on Workload Characterization (IISWC)*. DOI:http://dx.doi.org/10.1109/IISWC.2013.6704677

Erik Tomusk, Christophe Dubach, and Michael O'Boyle. 2015a. Diversity: A Design Goal for Heterogeneous Processors. *Computer Architecture Letters* (2015). DOI:http://dx.doi.org/10.1109/LCA.2015.2499739

Erik Tomusk, Christophe Dubach, and Michael O'boyle. 2015b. Four Metrics to Evaluate Heterogeneous Multicores. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 4 (Nov. 2015). DOI:http://dx.doi.org/10.1145/2829950

Yatish Turakhia, Bharathwaj Raghunathan, Siddharth Garg, and Diana Marculescu. 2013. HaDeS: Architectural synthesis for heterogeneous dark silicon chip multi-processors. In *Design Automation Conference (DAC)*.

Kenzo Van Craeynest and Lieven Eeckhout. 2013. Understanding fundamental design choices in single-ISA heterogeneous multicore architectures. *ACM Transactions on Architecture and Code Optimization (TACO)* 9, 4 (Jan. 2013). DOI:http://dx.doi.org/10.1145/2400682.2400691

Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. 2010. Conservation Cores: Reducing the Energy of Mature Computations. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. DOI:http://dx.doi.org/10.1145/1736020.1736044

Ganesh Venkatesh, Jack Sampson, Nathan Goulding-Hotta, Sravanthi Kota Venkata, Michael Bedford Taylor, and Steven Swanson. 2011. QsCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores. In *International Symposium on Microarchitecture (MICRO)*. DOI:http://dx.doi.org/10.1145/2155620.2155640

Richard Vuduc, James W. Demmel, and Jeff Bilmes. 2004. Statistical models for empirical search-based performance tuning. *International Journal of High Performance Computing Applications* 18, 1 (Feb. 2004). DOI:http://dx.doi.org/10.1177/1094342004041293

Sam Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *International Symposium on High Performance Computer Architecture (HPCA)*. 577–589. DOI:http://dx.doi.org/10.1109/HPCA.2015.7056064

Yuhao Zhu and Vijay Janapa Reddi. 2013. High-performance and energy-efficient mobile web browsing on big/little systems. In *International Symposium on High Performance Computer Architecture (HPCA2013)*. DOI:http://dx.doi.org/10.1109/HPCA.2013.6522303

Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (Nov. 1999). DOI:http://dx.doi.org/10.1109/4235.797969